JIVE UniBoard Correlator Memo 11:
Timing and synchronization

Harro Verkouter

September 26 2012

Background

The JIVE UniBoard Correlator (JUC) correlates VDIF formatted streams of radio-astronomical data from its memory buffers. During correlation corrections must be applied by the FPGAs to compensate for the rotation of the earth, the delay tracking. This compensation is time-variable and more specifically, the compensations must be applied at the correct time to the data.

The data and coefficients are sent to the JUC using UDP over IPv4, albeit over different, independent, network paths.

The JUC itself runs asynchronously; it has no concept of time or rate of time. At least not where data processing is concerned – data will be processed at whatever data rate the hardware can do irrespective of sample rate of the data. To this effect the architecture of the JUC is data driven processing: the availability of data sets the correlator in motion.

This memo discusses the method used to synchronize all parties involved: the JUC FPGAs and both the data and model coefficient senders. At the end of the memo a dedicated section describes all the complicating factors.

General description of the JUC synchronization mechanism

It was decided to make the JUC run on a one-second basis. The correlator control software (CCS) will be the principal synchronizing entity in the correlator system. Below the synchronizing system is described, intentionally leaving out many of the configuration details.

After initial set up and configuration correlation can start. CCS instructs data senders to send data from the first UT second of the time range which needs to be correlated. At the same time CCS instructs the model coefficient senders to load the coefficients for that same UT second.

CCS waits until all senders have acknowledged that they are finished, at which point CCS will inform the JUC that a second's worth of data can be processed by writing to a specific register.

The JUC holds a number of one-second data buffers and indicates in a register if there is another one free. If so, CCS can instruct the data- and model senders to send the next UT second of the time range to correlate. This repeats until there isn't a free buffer any more, after which CCS will start to poll the register to wait for a buffer to clear.

In this fashion CCS can step through the data, synchronizing all components involved.

Dealing with arbitrary integration time/spectral resolution

The JUC is an FX correlator.

This implies that for *N* point spectral resolution JUC first reads 2**N* samples from the input buffers, fast fourier transforms (FFT) them and only then sends the resulting FFT segment off for correlation.

An integral number of these individually correlated FFT segments can be integrated to create an integration. The correlator always processes a full integration at a time during which the process cannot be halted; the correlation process can only be put on hold at integration boundaries.

If CCS allows an arbitrary integral number of FFT segments to be integrated it cannot be guaranteed that integration boundaries and UT second boundaries always coincide.

If they don't, this causes less than the necessary amount of FFT segments remaining in a buffer, meaning that the integration cannot be processed, i.e. the correlator must stop until more bits, specifically those of the next UT second, have come in.

This, in turn, means that, technically, the current buffer cannot be marked as 'processed' because there is data left in there which is, well, unprocessed[1].

To this effect the buffers of JUC have been dimensioned to hold at least four seconds worth of data at 1024Mbps sampling rate (the highest rate it is meant to process). Theoretically three seconds should be enough (in that case there is always a free buffer to write data in) but four is easier to work with in hardware.

A direct consequence is that the acknowledgement of "there is one UT second's worth of data" is not enough; potentially not the whole second can be processed. CCS knows both the data sampling rate and the integration length. CCS can therefore predict, for each UT second, how many whole integrations can be processed.

To this effect CCS will write a number indicating how many whole integrations can be processed without reaching into the next UT second buffer to the JUC synchronizing device.

Another issue to take into account

is that due to delay compensation samples may actually have to be read out of the next UT second – this cannot be easily predicted when this happens.

Enlarging the amount of buffers to four seconds makes this easy to solve: only start processing if there are two seconds worth of data present in the buffers and stop if there is one second or less worth of data available. This would pose an issue at the end of the time range and cause the last second to be lost. However, CCS could easily write one extra, 'fake', acknowledgement to the JUC, forcing it to process the last remaining integrations from the last second of the correlated time range and then stop.

---

[1] see the section 'Complicating factors' below for why this is not possible

Synchronization hardware support

The JUC features a special bit of VHDL on front node 0 (FN0) to implement the synchronization device. Conceptually it may be thought of an integer first-in first-out device (FIFO) with a programmable depth, i.e. a programmable amount of free positions in it.

The FIFO has two ends: a read end, located in the VHDL of the correlator and a write end, exposed to CCS.

The FIFO also exposes two conditions: *empty* and *full*. The *full* condition is triggered when as many numbers are written to the FIFO as its depth is, without the read end taking any numbers off. The *empty* condition is raised if the FIFO fill level falls below a set lower limit.

The *empty* condition is exposed to the correlator in FN0, the *full* condition is exposed to CCS.

The FIFO will be empty initially. After the JUC has been configured the and told to start, the correlator firmware will wait for the empty condition to clear. If it clears, the firmware reads out the actual 32-bit number from the FIFO. This number will be interpreted as the number of whole integrations that can be processed from the data in the buffers.

NOTE: Contrary to normal FIFOs the JUC synchronization FIFO will only clear the *empty* condition if 'more than one' (rather than 'more than zero') elements are present[2].

CCS will have to do the dead-reckoning, keeping track of how many bits are still left in a particular buffer's data and adjust the amount of integrations accordingly if this amount grows beyond the amount of data required for an integration.

Example

Assume the integration time is seven FFT periods and there are 19 FFT periods per UT second[3].

After sending the first UT second of data two whole integrations can be processed (2x7 = 14 FFT segments necessary for these), leaving 5 FFT segments unprocessed. After finishing sending data and model, CCS writes the value of '2' into the FIFO. Because there is only one element in the FIFO yet, the *empty* condition will not clear.

The FIFO will also not register *full* yet: there are three more positions available. CCS starts sending the next UT second's worth of data. In total there are now 5 (left over from previous UT second) + 19 (from this UT second) = 24 FFT segments available. CCS writes the value of '3' into the FIFO because 3x7 = 21 FFT segments necessary.

Now the FIFO isn't *empty* anymore. JUC reads the value of '2' from the FIFO (the first number written) and will process two full integrations.

When CCS adds another UT second's worth of data to the buffers there are now 3 (leftover from previous) + 19 (next UT second) = 22 FFT segments available so CCS writes once more the value of '3'; three more integrations can be processed.

JUC now reads the second value from the FIFO (the first '3') and processes three full integrations. Etc.

---

[2] see 'Dealing with arbitrary integration time/spectral resolution' above

[3] if the amount of FFT segments per second is not integral the accounting can also be done in units of samples without affecting the procedure

The synchronization hardware is only present in FN0 and the JUC firmware will use the UniBoard INT A/B inter-fpga-node communication mechanism to distribute the FIFO *empty* condition from FN0 to the other three front nodes, effectively making them slave nodes to FN0.

In this way both CCS and JUC only have to consider the logic of dealing with one synchronization point for (a) the software and (b) all of the hardware, making everyone's life easier.

Complicating factors

There are a number of synchronization points that need to be realized for a succesfull correlation. This section briefly mentions those pro memori and only broadly tries to explain why it is important, in the hope it will serve as a means to better understand why the timing and synchronization has been chosen as it stands.

Correlation can only start if data from the same time range is present in all buffers across all participating front nodes. Without proper external synchronization mechanism this would require advanced communication between the front nodes, which is both vulnerable and undesirable to code this in VHDL.

The correlator correlates at an unknown speed, in general not processing the data at its sampling rate, making it impossible to make assumptions about the actual data processing rate.

Each front node has a limited amount of memory available for buffering data for a number of stations and subbands. These buffers are circular buffers meaning that the memory will be overwritten by newer data if no effective protection agains this happening is implemented.

Data input into the board is done via the 10Gbit ethernet ports connected to the four front nodes of the UniBoard using UDP. The UDP protocol has no feedback mechanism about socket buffer fill levels so a data sender is never told to stop sending, as is the case with TCP.

Sometimes a station will malfunction for some time, the network path to the station may be broken or the disk on which a station's data is recorded may fail. Some stations start observing late or finish early. This implies that it cannot be guaranteed that at all times all data for all stations is even available. This seems to contradict the first point mentioned but as long as missing data is flagged as invalid, the first point may be weakened to read 'correlation may start when *someone* is quite sure that as much data as available for that time range is present in the buffers'.

When data of a particular time range is correlated the model coefficients specific to that time range need to be present in the front nodes as well. The model coefficients are also sent via UDP albeit via a completely different path, meaning there is no synchronization whatsoever between data- and model senders.

All points combined implicate that an external feedback mechanism is required – both over- and underestimating of the actual data processing speeds will be desastrous for the correlator process.